Optimizing and Developing Non-CPU Device Power Management by DEVFREQ

Chanwoo Choi
Samsung Electronics
Contents

• DEVFREQ?
  – How To Add Devfreq Driver and Devfreq Governor
  – Sysfs Interface

• Collaboration with other Frameworks

• Simple Profiling and Performance Tuning Point

• Use-Case in Mainline Kernel

• Weakness of DEVFREQ and Further TODO
**Performance Demand**
- High-Quality image generated by GPU
- Data Transfer within deadline via Memory Bus
- Low Latency for accessing Storage

**Power-Consumption Requirement**
- Increase the battery capacity continuously

**Need to support ¹DVFS for Non-CPU Power-management**

**Provide power-management mechanism** for Non-CPU device to **keep balance** between Performance and Power
DEVFREQ with System Status

Busy Status
- CPUFREQ

Idle Status
- CPUIDLE
- CPU HOTPLUG
- Runtime PM (pm_runtime)
- Power Domain (genpd)

Suspend Status
- Suspend-to-RAM

Non-CPU Device
- GPU
- Memory Controller/Bus
- Storage
- Other Non-CPU devices
DEVFREQ Internal Module

Governor
- Get Device Status to get current device load
- Manage lifecycle of governor
- performance
- powersave
- userspace
- simple_ondemand
- passive
- Device Own Governor

Core
- Timer for checking current device load
- Adjust Final Freq to consider various requirement

Sysfs
- to show or store devfreq value

Suspending
- to suspend devfreq device and stop/start governor

PM QoS
- Governor Result
- OPP

User by Sysfs
- Thermal
- Interconnect
DEVFREQ relation with External Framework

1. SMCCC (Secure Monitor Call Calling Convention)
How To Add Devfreq Driver & How To Add Devfreq Governor
DEVFREQ Device Driver

• Support DVFS by controlling clock and voltage according to device status

• The kind of DEVFREQ Driver
  – **GPU** like ARM Panfrost/Lima, MSM GPU
  – **Memory Bus** like AMBA AXI Bus
  – **Memory Controller** like DMC (Dynamic Memory Controller)
  – **Storage** like UFS (Universal Flash Storage)
  – **L2 Cache**
  – And so on
Step to Add DEVFREQ Driver

1. Initialize ‘struct devfreq_dev_profile’ for device profile
   - Initialize ‘polling_ms’ of timer period and ‘timer’ is either deferrable or delayed timer.
   - Implement ‘target’ function to set the next frequency/voltage
   - Implement ‘get_dev_status’ function to get current device status

2. Get clock/regulator and OPP table from DeviceTree
   - devm_clk_get(), devm_regulator_get(), dev_pm_opp_of_add_table() or other OPP helper functions

3. Choose governor
   - simple_ondemand, userspace and others.

4. Add devfreq device
   - devm_devfreq_add_device(device, devfreq_dev_profile, governor, data)

5. (optional) Register OPP notifier
   - devm_devfreq_register_opp_notifier(device, devfreq)
# How to Add DEVFREQ Driver - struct devfreq_dev_profile

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Mandatory or Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>initial_freq</td>
<td>Initial frequency.</td>
<td>Optional</td>
</tr>
<tr>
<td>polling_ms</td>
<td>Polling interval for timer. If 0, disable polling. The unit is millisecond (ms).</td>
<td>Optional</td>
</tr>
<tr>
<td>timer</td>
<td>Timer type is either deferrable or delayed timer. The default value is deferrable timer.</td>
<td>Optional</td>
</tr>
<tr>
<td>up_threshold</td>
<td>If the load is over this value, the frequency jumps. Valid value = 0 to 100. Default value is 90. downdifferential &lt; upthreshold must hold.</td>
<td>Optional</td>
</tr>
<tr>
<td>down_differential</td>
<td>If the load is under upthreshold - downdifferential, the frequency downs. Valid value = 0 to 100. Default value is 5. downdifferential &lt; upthreshold must hold.</td>
<td>Optional</td>
</tr>
<tr>
<td>(*target)</td>
<td>Set operating frequency decided by devfreq core with both governo and PM QoS request</td>
<td>Mandatory</td>
</tr>
<tr>
<td>(*get_dev_status)</td>
<td>Return the current load of devfreq device. The result is used for deciding the next frequency by governor.</td>
<td>Optional</td>
</tr>
<tr>
<td>(*get_cur_freq)</td>
<td>Return the current correct frequency.</td>
<td>Optional</td>
</tr>
<tr>
<td>(*exit)</td>
<td>Exit the devfreq device.</td>
<td>Optional</td>
</tr>
</tbody>
</table>
DEVFREQ Governor

- Decide proper frequency by governor algorithm
- User can add the own device governor

Governor
  - simple_ondemand
  - performance
  - powersave
  - userspace
  - passive

Device Own Governor
  - tegra_actmon
Step to Add DEVFREQ Governor

1. Initialize ‘struct devfreq_governor’ for governor
   - Initialize ‘name’ of governor name.
   - Implement ‘get_target_freq’ function to get next frequency decided by governor algorithm.
   - Implement ‘event_handler’ function to handle the governor event for governor lifecycle.

2. Add devfreq governor
   - devfreq_add_governor(devfreq_governor);

3. The devfreq governor will be used by devfreq drivers.
## How to Add DEVFREQ Governor - struct devfreq_governor

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Mandatory or Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Governor name like “simple_ondemand”, “performance”.</td>
<td>Mandatory</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>attr</td>
<td><strong>Governor sysfs attribute flag.</strong> Basically, common sysfs attributes are added to devfreq class and need to initialize the following flags for using non-general sysfs attributes.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- DEVFREQ_GOV_ATTR_POLLING_INTERVAL : polling_interval</td>
<td>Optional</td>
</tr>
<tr>
<td></td>
<td>- DEVFREQ_GOV_ATTR_TIMER : timer</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- DEVFREQ_GOV_ATTR_UP_THRESHOLD : up_threshold</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- DEVFREQ_GOV_ATTR_DOWN_DIFF : down_differential</td>
<td></td>
</tr>
<tr>
<td>flag</td>
<td><strong>Governor feature flag</strong></td>
<td>Optional</td>
</tr>
<tr>
<td></td>
<td>- DEVFREQ_GOV_FLAG_IMMUTABLE : If set, this governor is never changeable to others.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- DEVFREQ_GOV_FLAG_IRQ_DRIVEN : If set, this governor is working with irq instead of timer.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(*get_target_freq)</td>
<td><strong>Return the desired operating frequency for the device according to governor algorithm.</strong></td>
<td>Mandatory</td>
</tr>
<tr>
<td>(*event_handler)</td>
<td><strong>Callback for devfreq core to notify events to governors.</strong></td>
<td>Mandatory</td>
</tr>
<tr>
<td></td>
<td>- DEVFREQ_GOV_START : When governor start</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- DEVFREQ_GOV_STOP : When governor stop</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- DEVFREQ_GOV_UPDATE_INTERVAL : When timer interval is updated via sysfs</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- DEVFREQ_GOV_SUSPEND : When governor suspend</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- DEVFREQ_GOV_RESUME : When governor resume</td>
<td></td>
</tr>
</tbody>
</table>
Example to Add Device Own Governor

“tegra_actmon” governor

in drivers/devfreq/tegra30_devfreq.c

static struct devfreq_governor tegra_devfreq_governor = {
    .name = "tegra_actmon",
    .attr = DEVFREQ_GOV_ATTR_POLLING_INTERVAL,
    .flag = DEVFREQ_GOV_FLAG_IMMUTABLE |
             DEVFREQ_GOV_FLAG_IRQ_DRIVEN,
    .get_target_freq = tegra_governor_get_target,
    .event_handler = tegra_governor_event_handler,
};

‘poling_interval’ sysfs is used for Tegra ACTMON h/w period setting.

‘_FLAG_IMMUTABLE’ means that if device used ‘tegra_actmon’ governor, it cannot change to other governors.

‘_IRQ_DRIVEN’ means that it is based on interrupt method instead of timer.

Return the desired operating frequency according to Tegra ACTMON governor algorithm.

In Summary,

• Immutable and Interrupt method for sampling
• Use Tegra ACTMON Governor instead of default devfreq governors
DEVFREQ Driver and Governor Behavior

**DEVFREQ Governor**
- Manage lifecycle of governor
- If governor is immutable, it cannot be changed.
  - sysfs: governor
  - sysfs: up_threshold
  - sysfs: down_differential

**Device Governor**
- Get Device Status

**DEVFREQ Core**
- sysfs: min_freq
- sysfs: max_freq
- User via sysfs
- Interconnect
- Device
- OPP
- PM QoS Request
  - dev_pm_qos_update_request()

**Timer**
- devfreq_governor->event_handler()
- sysfs: polling_interval
- sysfs: timer

**DEVFREQ Device Driver**
- devfreq_dev_profile->get_dev_status()

**Adjust Final Freq**
- devfreq_governor->get_target_freq()
- 1. Re-run Timer
- 2. Get target freq from governor
- 3. Get current device load
- 4. Return target frequency from Governor
- 5. Set final frequency

Additional Features:
- DEVFREQ_GOV_START
- DEVFREQ_GOV_STOP
- DEVFREQ_GOV_SUSPEND
- DEVFREQ_GOV_RESUME
- DEVFREQ_GOV_UPDATE_INTERVAL

- Manage lifecycle of governor
- User via sysfs
- Thermal
- Interconnect
- Device
- OPP
- Set Min and Max Freq
- Enable and Disable Freq
- opp_disable
- opp_enable

-opp_disable
- opp_enable
Simple_ondemand Governor Behavior

in drivers/devfreq/governor_simple_ondemand.c
static struct devfreq_governor devfreq_simple_ondemand = {
    .name = DEVFREQ_GOV_SIMPLE_ONDEMAND,
    .attr = DEVFREQ_GOV_ATTR_POLLING_INTERVAL
        | DEVFREQ_GOV_ATTR_TIMER
        | DEVFREQ_GOV_ATTR_UP_THRESHOLD
        | DEVFREQ_GOV_ATTR_DOWN_DIFF,
    .get_target_freq = devfreq_simple_ondemand_func,
    .event_handler = devfreq_simple_ondemand_handler,
};
Passive Governor

• Depend on behavior of parent device such as other devfreq device or CPU.

• Pattern of using passive governor
  – Two devices share the same power source.
Example of Passive Governor

- **Memory Bus Device**
  in Samsung Exynos5422 SoC
  - VDD_INT regulator provides power to 15 AMBA AXI Bus device.

- **Step to change freq/voltage**
  - Decide next freq/voltage on WCORE device.
    - User can change the governor of WCORE device via sysfs and then passive device freq will be changed.
  - If next freq is higher than previous freq,
    - Change WCORE’s freq & voltage
    - Change freq of 15 AMBA AXI Bus
  - If next freq is less than previous freq,
    - Change freq of 15 AMBA AXI bus
    - Change WCORE’s freq & voltage

---

1. AMBA (Advanced Microcontroller Bus Architecture) AXI (Advanced eXtensible Interface)
Sysfs Interface
# Common Sysfs Interface for Devfreq Class

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>RW</th>
</tr>
</thead>
<tbody>
<tr>
<td>governor</td>
<td>Show and store the <strong>current governor name</strong></td>
<td>RW</td>
</tr>
<tr>
<td>available_governor</td>
<td>Show the <strong>available governor list</strong></td>
<td>RO</td>
</tr>
<tr>
<td>available_frequencies</td>
<td>Show the <strong>available frequencies</strong></td>
<td>RW</td>
</tr>
<tr>
<td>target_freq</td>
<td>Show the <strong>current frequency</strong> of the one of OPP table</td>
<td>RO</td>
</tr>
<tr>
<td>cur_freq</td>
<td>Show the current frequency of hardware clock rate</td>
<td>RO</td>
</tr>
<tr>
<td></td>
<td>if <code>get_cur_freq()</code> is implemented by devfreq driver.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>If <code>get_cur_freq()</code> is not implemented, it is same with <code>taget_freq</code>.</td>
<td></td>
</tr>
<tr>
<td>min_freq</td>
<td>Show and store the <strong>minimum frequency</strong></td>
<td>RW</td>
</tr>
<tr>
<td>max_freq</td>
<td>Show and store the <strong>maximum frequency</strong></td>
<td>RW</td>
</tr>
<tr>
<td>trans_stat</td>
<td>Show the <strong>frequency transition statistics</strong> and <strong>time in state</strong></td>
<td>RW</td>
</tr>
<tr>
<td></td>
<td>To reset the statistics as following:</td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>echo 0 &gt; /sys/class/devfreq/[dev name]/trans_stat</code></td>
<td></td>
</tr>
</tbody>
</table>
Non-Common Sysfs Interface for Devfreq Governor

- Each governor is able to choose the following sysfs nodes if it is needed

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>RW</th>
<th>Use-case</th>
</tr>
</thead>
<tbody>
<tr>
<td>timer</td>
<td>Show and store the <strong>timer type (deferrable or delayed)</strong></td>
<td>RW</td>
<td><strong>simple_ondemand</strong></td>
</tr>
<tr>
<td></td>
<td>- DEVFREQ_GOV_ATTR_TIMER</td>
<td></td>
<td></td>
</tr>
<tr>
<td>polling_interval</td>
<td>Show and store the <strong>polling interval</strong></td>
<td>RW</td>
<td><strong>tegra_actmon</strong></td>
</tr>
<tr>
<td></td>
<td>- DEVFREQ_GOV_ATTR_POLLING_INTERVAL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>up_threshold</td>
<td>Show and store <strong>up_threshold</strong> tuning point</td>
<td>RW</td>
<td><strong>simple_ondemand</strong></td>
</tr>
<tr>
<td></td>
<td>- flag name: DEVFREQ_GOV_UP_THRESHOLD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>down_differential</td>
<td>Show and store <strong>down_differential</strong> tuning point</td>
<td>RW</td>
<td><strong>simple_ondemand</strong></td>
</tr>
<tr>
<td></td>
<td>- flag name: DEVFREQ_GOV_DOWN_DIFF</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## DEVFREQ Governor of both Sysfs and Feature Flags

<table>
<thead>
<tr>
<th>sysfs node</th>
<th>governor</th>
<th>simple _ondemand</th>
<th>peformance</th>
<th>powersave</th>
<th>userspace</th>
<th>passive</th>
<th>tegra30_actmon</th>
</tr>
</thead>
<tbody>
<tr>
<td>Common Sysfs Interface for devfreq class</td>
<td>governor</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
</tr>
<tr>
<td>available_governors</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
</tr>
<tr>
<td>available_frequencies</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
</tr>
<tr>
<td>cur_freq</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
</tr>
<tr>
<td>target_freq</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
</tr>
<tr>
<td>min_freq</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
</tr>
<tr>
<td>max_freq</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
</tr>
<tr>
<td>trans_stat</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
</tr>
<tr>
<td>Non-common Sysfs Interface for specific governor</td>
<td>polling_interval</td>
<td>O</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>timer</td>
<td>O</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>up_threshold</td>
<td>O</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>down_differential</td>
<td>O</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Governor feature for specific governor</td>
<td>immutable</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>O</td>
<td>O</td>
</tr>
<tr>
<td>interrupt_driven</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>O</td>
</tr>
</tbody>
</table>

*Note: (polling based on timer) (polling based on h/w irq)*
Collaboration with other Frameworks
- OPP
- PM QoS
- Interconnect
- Thermal
What is Collaboration Purpose with other Framework?

- **Boosting or Constraining Frequency**
  - Change the frequency range according to mode
    - *ie.* Performance, Optimized, Powersave and Ultra-Powersave mode
  - Boosting for preventing performance drop
  - Constraining for preventing either high-temperature or misuse power

---

**Diagram Explanation**

- **User via sysfs**: Set min and max frequency
- **Thermal**: dev_pm_qos_update_request()
- **Interconnect**: opp_disable, opp_enable
- **Device directly**: DEVFREQ Governor
- **Available 5 Frequency Level**
  - 1200 MHz
  - 1100 MHz
  - 1000 MHz
  - 900 MHz
  - 800 MHz
  - 700 MHz
  - 600 MHz
  - 400 MHz
- **DEVFREQ Core**: Final Freq
- **DEVFREQ Devcie Driver**: Target Freq

---

**Legend**

- DEVFREQ: Device Frequency
- Governor: Frequency Governor
- Core: Frequency Core
- Devcie Driver: Device Driver
OPP (Operating Performance Points) with DEVFREQ

- OPP is mandatory to support DVFS with frequency and voltage.
- OPP provides helper function to get OPP info from devicetree
  - dev_pm_opp_of_add_table()
  - dev_pm_opp_of_remove_table()
- Handle clock and regulator by OPP helper function
  - dev_pm_opp_set_rate()
  - dev_pm_opp_set_regulators()
  - dev_pm_opp_put_regulators()
OPP - Enable & Disable Each Frequency

- Enable and Disable OPP entries
  - dev_pm_opp_disable(dev, 20000000)
  - dev_pm_opp_disable(dev, 134000000)
  - dev_pm_opp_disable(dev, 100000000)
  - dev_pm_opp_enable(dev, 134000000)

in arch/arm/boot/dts/exynos3250.c

bus_dmc_opp_table: opp_table1 {
  compatible = "operating-points-v2";

  opp-50000000 {
    opp-hz = /bits/ 64 <50000000>;
    opp-microvolt = <800000>;
  }

  opp-100000000 {
    opp-hz = /bits/ 64 <100000000>;
    opp-microvolt = <800000>;
  }

  opp-134000000 {
    opp-hz = /bits/ 64 <134000000>;
    opp-microvolt = <800000>;
  }

  opp-200000000 {
    opp-hz = /bits/ 64 <200000000>;
    opp-microvolt = <825000>;
  }

  opp-400000000 {
    opp-hz = /bits/ 64 <400000000>;
    opp-microvolt = <875000>;
  }
};

bus_dmc: bus_dmc {
  compatible = "Samsung,exynos-bus";
  clocks = <&cmu_dmc CLK_DIV_DMC>;
  clock-names = "bus";
  operating-points-v2 = <&bus_dmc_opp_table>;
}
PMQoS used by DEVFREQ

- Set min and max frequency to guarantee device’s demand
  - DEV_PM_QOS_MIN_FREQUENCY
  - DEV_PM_QOS_MAX_FREQUENCY

- Example,
  - PM QoS Request
    1. dev_pm_qos_add_request(device, qos_request, DEV_PM_QOS_MIN_FREQUENCY)
    2. dev_pm_qos_request_active(qos_request)
    3. dev_pm_qos_update_request(qos_request, frequency)
  - Read PM QoS Requests
    1. dev_pm_qos_read_value(dev, DEV_PM_QOS_MIN_FREQUENCY)
  - PM QoS Release
    1. dev_pm_qos_update_request(qos_request, 0)
    2. dev_pm_qos_remove_request(qos_request)
Interconnect with DEVFREQ

- Interconnect framework control the setting of the ‘interconnects on an SoC’ like memory controller and data bus.

- Two framework might be connected through PM QoS interface for guaranting performance.

<table>
<thead>
<tr>
<th>DEVFREQ Device Driver</th>
<th>Bridge by PM QoS Interface</th>
<th>INTERCONNECT Device Driver</th>
</tr>
</thead>
<tbody>
<tr>
<td>drivers/devfreq/imx-bus.c</td>
<td>PM QoS</td>
<td>drivers/interconnect/imx/imx.c</td>
</tr>
<tr>
<td>drivers/devfreq/imx8m-ddrc.c</td>
<td>DEV_PM_QOS_MIN_FREQUENCY</td>
<td>drivers/interconnect/imx/imx8mq.c</td>
</tr>
<tr>
<td>IMX SoC</td>
<td></td>
<td>drivers/interconnect/imx/imx8mm.c</td>
</tr>
<tr>
<td></td>
<td></td>
<td>drivers/interconnect/imx/imx8mn.c</td>
</tr>
<tr>
<td>drivers/devfreq/exynos-bus.c</td>
<td>PM QoS</td>
<td>drivers/interconnect/exynos/exynos.c</td>
</tr>
<tr>
<td></td>
<td>DEV_PM_QOS_MIN_FREQUENCY</td>
<td>(But, not yet merged and under review)</td>
</tr>
</tbody>
</table>

Example of connection between DEVFREQ and Interconnect Device Driver
Example of NXP I.MX SoC

**DEVFREQ**
- drivers/devfreq/imx-bus.c
- drivers/devfreq/imx8-m-ddrc.c

**PM QoS**
- DEV_PM_QOS_MIN_FREQUENCY
- Register child platform_device for interconnect device
- platform_device_register_data(parent device of imx-bus, ...)

**INTERCONNECT**
- Probe of interconnect device/driver
- drivers/interconnect/imx/imx.c
- Create interconnect node
- Platform driver for interconnect device:
  - drivers/interconnect/imx/imx8mq.c
  - drivers/interconnect/imx/imx8mm.c
  - drivers/interconnect/imx/imx8mn.c
Thermal with DEVFREQ

- Register devfreq device as a cooling device
  - drivers/thermal/devfreq_cooling.c

- Basically, adjust frequency by using fixed trip-points defined in devicetree
  - step_wise thermal governor

- More Advanced than fixed trip-points method, adjust frequency with IPA (Intelligent Power Allocator) governor using EM (Energy Model).
Thermal IPA and EM with DEVFREQ

CPU Device

- CPUFREQ
- EAS (Energy Aware Scheduler)
- Energy Model

Non-CPU Device

- DEVFREQ
- IPA (Intelligent Power Allocator)
- Non-CPU Device (e.g., GPU)
Simple Profiling & Performance Tuning Point

- sysfs
- debugfs
- tracepoint
- genpd, pm_runtime
Simple Profiling

- **Debugfs**
  - devfreq_summary

- **Sysfs (Sampling)**
  - min_freq, cur_freq, max_freq and trans_stat sysfs interface

- **Tracepoint (Tracing)**
  - devfreq, thermal, power

- **Device power status** (active or suspended)
  - Generic Power Domain and Runtime PM
## Simple Profiling - Debugfs Interface

- **/sys/kernel/debug/devfreq/devfreq_summary**
  - Test device: Odroid-XU3 (Samsung Exynos5422 SoC)
    - 17 Non-CPU devices
    - 1 GPU (11800000.gpu) / `simple_ondemand` / 177MHz ~ 600MHz
    - 1 Memory Controller (10c20000.memory-controller) / `performance` / 165MHz~825Mz
    - 1 AMBA AXI Bus (soc:bus_wcore) / `simple_ondemand` / 88.7MHz ~ 532MHz
    - 14 AMBA AXI Bus with 'soc:bus_wcore' parent device / passive

![Devfreq Summary Table](image-url)
Simple Profiling - Sysfs Interface

• **min_freq, max_freq and cur_freq**  
  - How to control them  
    • echo [available frequency] > /sys/class/devfreq/[dev name]/min_freq and read it  
    • echo [available frequency] > /sys/class/devfreq/[dev name]/max_freq and read it  
    • cat > /sys/class/devfreq/[dev name]/cur_freq  
  - How to use them for profiling  
    • Make simple shell script to print the frequency periodically.

• **trans_stat**  
  - Show transition statistics and time in each frequency  
  - How to control them  
    • echo 0 > /sys/class/devfreq/[dev name]/trans_stat and read it
Simple Profiling - Sysfs ‘trans_stat’

<table>
<thead>
<tr>
<th>From</th>
<th>To</th>
<th>time(ms)</th>
<th>165000000</th>
<th>206000000</th>
<th>275000000</th>
<th>413000000</th>
<th>543000000</th>
<th>633000000</th>
<th>728000000</th>
<th>825000000</th>
</tr>
</thead>
<tbody>
<tr>
<td>165000000</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>206000000</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>275000000</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>413000000</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>5619</td>
<td>620750</td>
</tr>
<tr>
<td>543000000</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>857</td>
<td>95100</td>
</tr>
<tr>
<td>633000000</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>728000000</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>*825000000</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>5619</td>
<td>857</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>861580</td>
</tr>
</tbody>
</table>

Total transition: 12958

- **How to use it for performance profiling**
  1. Try to tune and optimize the your code
  2. **Reset trans_stat** by ‘echo 0 > /sys/class/devfreq/[dev name]/trans_stat’
  3. **Read ‘time_in_state’** by ‘cat /sys/class/devfreq/[dev name]/trans_stat’
  4. **Execute benchmark tool**
  5. Read ‘time_in_state’
  6. **Calculate diff ‘time_in_state’** between before and after benchmark tool
Simple Profiling - Tracepoint

**DEVFREQ**
- Track devfreq behavior when frequency change and monitoring
  - /sys/kernel/debug/tracing/event/devfreq/devfreq_frequency
  - /sys/kernel/debug/tracing/event/devfreq/devfreq_monitor

**Thermal**
- Track throttling or un-throttling point due to high temperature
  - /sys/kernel/debug/tracing/event/thermal/thermal_temperature
  - /sys/kernel/debug/tracing/event/thermal/thermal_zone_trip

**PM QoS**
- Track what request the pm qos of both minimum and maximum freq
  - /sys/kernel/debug/tracing/event/power/dev_pm_qos_update_request
Simple Profiling - Tracepoint Event

- **devfreq_monitor**
  - Show when device monitoring is executed by timer.
  - It is used to check how often it has been monitored.

- **devfreq_frequency**
  - Show frequency change point.
  - This is useful for determining whether or not the frequency has changed at the appropriate timing and checking the history of frequency change.

- **thermal_temperature**
  - Show temperature of thermal device like CPU, GPU

- **thermal_zone_trip**
  - Show trip point when arrived at the specific temperature (throttling or un-throttling)

- **dev_pm_qos_update_request**
  - Show qos request point with request value
Simple Profiling - Tracepoint

• How to use it for performance profiling
  1. Enable tracepoint of devfreq, PM QoS and thermal

  2. Enable tracepoint of performance-sensitive devices
     • DRM for display controller
       – if 60 fps is required, each vblank interrupt must happen within approximate 16 ms.
     • V4L2 for video playback
     • Storage access latency
Simple Profiling – Tracepoint Example

PM QoS request to Devfreq device

Check interval of each vblank irq

Frequency Change

Check throttling or not w/ thermal

(snip)
Simple Profiling - Device Status

- **Runtime PM**
  - `/sys/devices/platform/soc/[device name]/power/runtime_status`

- **Generic Power Domain**
  - `/sys/kernel/debug/pm_genpd/pm_genpd_summary`

![Diagram of CPU and Non-CPU Device Statuses](image)
Simple Profiling - Check device status via Debugfs

- Can check the device status as following:

```
$ cat /sys/kernel/debug/pm_genpd/pm_genpd_summary
```

<table>
<thead>
<tr>
<th>domain</th>
<th>status</th>
<th>children</th>
<th>runtime status</th>
</tr>
</thead>
<tbody>
<tr>
<td>/device</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CAM</td>
<td>off-0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MSC</td>
<td>off-0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MAU</td>
<td>on</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DISP</td>
<td>on</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>/device</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>/devices/platform/soc/10010000.clock-controller/exynos5-subcmu.3.auto</td>
<td>active</td>
</tr>
<tr>
<td></td>
<td></td>
<td>/devices/platform/soc/14650000.sysmmu</td>
<td>active</td>
</tr>
<tr>
<td></td>
<td></td>
<td>/devices/platform/soc/14640000.sysmmu</td>
<td>suspended</td>
</tr>
<tr>
<td></td>
<td></td>
<td>/devices/platform/soc/14680000.sysmmu</td>
<td>suspended</td>
</tr>
<tr>
<td></td>
<td></td>
<td>/devices/platform/soc/14450000.mixer</td>
<td>active</td>
</tr>
<tr>
<td></td>
<td></td>
<td>/devices/platform/soc/14530000.hdmi</td>
<td>active</td>
</tr>
<tr>
<td>G3D</td>
<td>on</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>/devices/platform/soc/10010000.clock-controller/exynos5-subcmu.2.auto</td>
<td>active</td>
</tr>
<tr>
<td></td>
<td></td>
<td>/devices/platform/soc/11800000.gpu</td>
<td>suspended</td>
</tr>
<tr>
<td></td>
<td></td>
<td>/devices/platform/soc/14530000.hdmi</td>
<td>active</td>
</tr>
</tbody>
</table>

Test device: Odroid-XU3 (Samsung Exynos5422 SoC)

The `devfreq_summary` shows ‘GPU’ devfreq device.

But, when try to print tracepoint for GPU, doesn’t work, even if GPU device uses `simple_ondemand` with timer.

Because GPU device status is suspended by runtime PM.
Support runtime PM for device to reduce power waste

- DEVFREQ provides the following governor helper function to control governor according to device power status

<table>
<thead>
<tr>
<th>Governor Helper Function</th>
<th>Governor Status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>devfreq_monitor_start(struct devfreq)</td>
<td>DEVFREQ_GOV_START</td>
<td>Start governor for devfreq device</td>
</tr>
<tr>
<td>devfreq_monitor_stop(struct devfreq)</td>
<td>DEVFREQ_GOV_STOP</td>
<td>Stop governor for devfreq device</td>
</tr>
<tr>
<td>devfreq_monitor_suspend(struct devfreq)</td>
<td>DEVFREQ_GOV_SUSPEND</td>
<td>Suspend governor for devfreq device. If this has ‘suspend-opp’ property in devicetree, set suspend frequency indicated by ‘suspend-opp’.</td>
</tr>
<tr>
<td>devfreq_monitor_resume(struct devfreq)</td>
<td>DEVFREQ_GOV_RESUME</td>
<td>Resume governor for devfreq device. If this has ‘suspend-opp’ property in devicetree, recover the last frequency</td>
</tr>
<tr>
<td>devfreq_update_interval(struct devfreq)</td>
<td>DEVFREQ_GOV_UPDATE_INTERVAL</td>
<td>Update polling interval by devfreq device driver instead of sysfs interface</td>
</tr>
</tbody>
</table>
Performance Tuning Point - Sysfs

• timer and polling_interval
  – echo (deferrable|delayed) > /sys/class/devfreq/[dev name]/timer
    • deferrable timer is not expired if CPU idle.
    • delayed timer doesn’t care CPU status.
  – echo [positive integer] > /sys/class/devfreq/[dev name]/polling_interval

• up_threshold and down_differential
  – echo [0-100] > /sys/class/devfreq/[dev name]/up_threshold
  – echo [0-100] > /sys/class/devfreq/[dev name]/down_differential
Performance Tuning Point - Deferrable vs. Delayed

- Difference between deferrable vs. delayed timer
  - Need CONFIG_HIGH_RES_TIMERS and CONFIG_NO_HZ for deferrable timer
Performance Tuning Point - Deferrable timer’s bad case

- In case of DMA without CPU operation, DMA operation transfer data between memory and device.

Diagram showing the relationship between device frequency, DMA operation, and performance drop.
Performance Tuning Point - Sysfs

• timer and polling_interval

<table>
<thead>
<tr>
<th>timer</th>
<th>polling_interval</th>
<th>Recommendation use-case But, it is not always true.</th>
</tr>
</thead>
<tbody>
<tr>
<td>deferrable</td>
<td>long</td>
<td>- The non-cpu device is related to CPU status</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Don’t want to wakeup CPU due to dev monitoring</td>
</tr>
<tr>
<td>deferrable</td>
<td>short</td>
<td>- Don’t want to wakeup CPU due to dev monitoring</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Need to react fastly on CPU busy</td>
</tr>
<tr>
<td>delayed</td>
<td>long</td>
<td>- The non-cpu device is less or not related to CPU status like DMA.</td>
</tr>
<tr>
<td>delayed</td>
<td>short</td>
<td>- The non-cpu device is less or not related to CPU status like DMA.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Need to react fastly always</td>
</tr>
</tbody>
</table>

• up_threshold and down_differential

<table>
<thead>
<tr>
<th>Freq Up Speed</th>
<th>Freq Down Speed</th>
<th>up_threshold</th>
<th>down_differential</th>
<th>Performance vs. Low Power</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fastly</td>
<td>Fastly</td>
<td>low</td>
<td>high</td>
<td>-</td>
</tr>
<tr>
<td>Fastly</td>
<td>Slowly</td>
<td>low</td>
<td>low</td>
<td>Highest Performance</td>
</tr>
<tr>
<td>Slowly</td>
<td>Fastly</td>
<td>high</td>
<td>high</td>
<td>-</td>
</tr>
<tr>
<td>Slowly</td>
<td>Slowly</td>
<td>high</td>
<td>low</td>
<td>Lowest Power</td>
</tr>
</tbody>
</table>
# Performance Tuning Point - Summary

<table>
<thead>
<tr>
<th>timer</th>
<th>polling_interval</th>
<th>up_threshold</th>
<th>down_differential</th>
<th>min_freq</th>
<th>max_freq</th>
<th>PM QoS Request</th>
<th>Recommendation use-case</th>
</tr>
</thead>
<tbody>
<tr>
<td>How often monitoring dev in accordance with CPU busy or idle</td>
<td>How often monitoring dev on CPU busy for reactivly</td>
<td>Frequency Up Speed for reactivity</td>
<td>Frequency Down Speed for reactivity</td>
<td>Frequency Boosting for high-performance</td>
<td>Resource Limiting for low-power or high-temperature</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Powersave Governor</td>
</tr>
<tr>
<td>deferrable</td>
<td>long</td>
<td>high</td>
<td>low</td>
<td>default</td>
<td>Lower max_freq close to min_freq</td>
<td>-</td>
<td>Lowest Power</td>
</tr>
<tr>
<td>deferrable</td>
<td>short</td>
<td>-</td>
<td>-</td>
<td>Non-Aggressive Use</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>deferrable</td>
<td>long</td>
<td>-</td>
<td>-</td>
<td>Aggressive Use</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>delayed</td>
<td>short</td>
<td>low</td>
<td>low</td>
<td>Aggressive Use</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>delayed</td>
<td>short</td>
<td>low</td>
<td>low</td>
<td>Higher min_freq close to max_freq</td>
<td>default</td>
<td>Aggressive use</td>
<td></td>
</tr>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Highest Performance</td>
</tr>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Performance Governor</td>
</tr>
</tbody>
</table>
Use-Case in Mainline Kernel
DEVFREQ Driver in Mainline Kernel (1/2)

- GPU
- ARM AMBA Bus
- DMC (Dynamic Memory Controller)
- UFS (Universal Flash Storage) Storage
- L2 Cache
  - Recently, mainline posted for Qualcomm Krait L2 Cache and under review.
## DEVFREQ Driver in Mainline Kernel (2/2)

<table>
<thead>
<tr>
<th>Device Type</th>
<th>Driver Path</th>
<th>SoC Vendor</th>
<th>Used Governor</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>GPU</strong></td>
<td>drivesr/gpu/drm/panfrost/panfrost_devfreq.c</td>
<td>ARM</td>
<td>simple_ondemand</td>
<td>Almost driver have been using simple_ondemand governor.</td>
</tr>
<tr>
<td></td>
<td>drivesr/gpu/drm/lima/lima_devfreq.c</td>
<td>ARM</td>
<td>simple_ondemand</td>
<td></td>
</tr>
<tr>
<td></td>
<td>drivesr/gpu/drm/msm/msm_gpu.c</td>
<td>Qualcomm</td>
<td>simple_ondemand</td>
<td></td>
</tr>
<tr>
<td>Memory Controller</td>
<td>drivers/memory/samsung/exynos5422-dmc.c</td>
<td>Samsung</td>
<td>simple_ondemand</td>
<td></td>
</tr>
<tr>
<td></td>
<td>driver/devfreq/imx8m-ddrc.c</td>
<td>NXP</td>
<td>simple_ondemand</td>
<td></td>
</tr>
<tr>
<td></td>
<td>drivers/devfreq/rk3399_dmc.c</td>
<td>Rockchip</td>
<td>simple_ondemand</td>
<td></td>
</tr>
<tr>
<td>Memory Data Bus</td>
<td>drivers/devfreq/exynos-bus.c</td>
<td>Samsung</td>
<td>simple_ondemand passive</td>
<td></td>
</tr>
<tr>
<td></td>
<td>drivers/devfreq/imx-bus.c</td>
<td>NXP</td>
<td>simple_ondemand</td>
<td></td>
</tr>
<tr>
<td>Storage (UFS)</td>
<td>drivers/scsi/ufs/ufshcd.c</td>
<td>Generic device</td>
<td>simple_ondemand</td>
<td></td>
</tr>
<tr>
<td>Specific SoC Device</td>
<td>drivers/devfreq/tegra30_devfreq.c</td>
<td>Nvidia</td>
<td>tegra_actmon</td>
<td></td>
</tr>
<tr>
<td></td>
<td>drivers/devfreq/tegra20_devfreq.c</td>
<td>Nvidia</td>
<td>tegra_actmon</td>
<td></td>
</tr>
</tbody>
</table>
Weakness of DEVFREQ & Further TODO
What are Weakness of DEVFREQ?

- **Too old governor** based on timer-based sampling method. Need new governor or method to monitor device for immediate response.
  - Schedutil governor of CPUFREQ framework

- **Too simply checking the device status at that time** without considering history and don’t expect future device status to prevent performance drop.
  - PELT (Per-Entity Load Tracking) of CPU scheduler
  - Ladder governor of CPUIDLE framework
Further ToDo

- Support ‘required-opp’ property of OPP to specify the correct pair between parent and passive device.
- Expand ‘passive’ governor depend on CPU Frequency
  - In the mainline, there are many requirement about this. But, it has not yet completed.
- For more immediate response, support kthread-based timer
  - DEVFREQ_TIMER_WQ_DEFERRABLE : Deferrrable work
  - DEVFREQ_TIMER_WQ_DELAYED : Delayed work
  - DEVFREQ_TIMER_KTHREAD : Kthread with SCHED_NORMAL
  - DEVFREQ_TIMER_KTHREAD_RT : Kthread with SCHED_FIFO
- Need kselfset for DEVFREQ device
Appendix

• ‘devfreq_frequency’ tracepoint patch (merged to devfreq.git)
  – [link](https://git.kernel.org/pub/scm/linux/kernel/git/chanwoo/linux.git/log/?h=devfreq-next)

• [v4,0/2] PM / devfreq: Add governor feature and attribute flag
  – [link](https://patchwork.kernel.org/project/linux-pm/cover/20201020030407.21047-1-cw00.choi@samsung.com/)

• up_threshold and down_differential patch (not yet posted, but can refer to it)
  – [link](https://git.kernel.org/pub/scm/linux/kernel/git/chanwoo/linux.git/log/?h=devfreq-testing)